M&G No. 50002.4USU1

METHOD AND SYSTEM FOR NETWORK LOAD BALANCING WITH A COMPOUND DATA STRUCTURE

Related Application

This application claims the benefit under 35 U.S.C. § 119(e) of U.S. Provisional Application No. 60/140,272, filed June 18, 1999.

Field of the Invention

The present invention relates to methods of implementing different types of data structures in database systems, and in particular, to a complex data structure for retrieving data from a database.

Background of the Invention

Data such as letters, numbers, and words are often maintained and stored in different types of data structures that complement their eventual with an application program. Since a List data structure enables the storing of data objects in an ordered set, this data structure is typically employed with an application program that often retrieves several sequentially related data objects at a time. For example, a multi-media player program usually generates streaming video from sequentially ordered data objects in a List data structure. Although retrieval times for large numbers of sequentially ordered or related data objects can be improved by the use of a List data structure, it may not always be the optimal data structure for retrieving individual and/or several non-sequential or unrelated data objects from a large set of data objects. It is well known that the maximum amount of time to locate an individual data object on a list increases as the number of data objects referenced by the list increase.

Another type of data structure is the Trie, which is often used with an application program that requests individual and/or several non-sequential or unrelated data objects. A Trie stores data in each transition between each node in a multi-level

data structure, rather than at the node itself. In this way, all of the transitions in the path between the root and each leaf represent the data object (key) and each transition between each node in the path is associated with a single character/number of the key. Since the nodes themselves are unlabeled, each transition between each node is labeled with a particular character/number. Also, the Trie data structure facilitates the calculation of a constant maximum amount of time to retrieve a stored data object based on the number of levels (nodes associated with the number of alphanumeric characters in the longest data object) in the Trie, e.g., $O(1)$. Thus, the maximum search time for a given number of levels in a Trie data structure remains relatively constant as the actual number of data objects stored in the data structure increases.

For example, a dictionary program often employs a Trie data structure to store words and provide relatively constant maximum search times as the number of words (but not their length) in the dictionary increase. In a Trie data structure for a dictionary program, every character in the alphabet (A through Z) is partitioned into individual and disjoint main level search nodes. Also, the total number of search node levels in a Trie data structure for a dictionary corresponds to the number of characters in the dictionary's longest possible word.

Although the List and Trie data structures can complement the operation of particular application programs, there are many other types of application programs that can generate requests for both individual/unrelated and sequential/related data objects. A facility that could employ a particular request generated by an application program to choose a type of data structure most suited to reference and retrieve requested data would be an improvement over the prior art. Also, a data structure that could provide access by one or more unique keys to the same data and enable the automatic removal of all references to deleted data would complement the operation of application programs that generate different types of requests.

A more complete appreciation of the invention and its improvements can be obtained by reference to the accompanying drawings, which are briefly summarized below, to the following detail description of presently preferred embodiments of the invention, and to the appended claims.

Summary of the Invention

In accordance with the invention, the above and other problems are solved by employing a plurality of data structures to optimize the retrieval of at least one data object over a network. Each data object is stored in a data store and each data object is separately referenced in each of the plurality of data structures. In response to a request for one data object, one of the plurality of data structures is automatically determined to be best suited to retrieve the one data object. The determined data structure is employed to locate and retrieve the one data object from the data store. Also, in response to a request for a plurality of related data objects, another one of the plurality of data structures is automatically determined to be best suited to retrieve the plurality of related data objects. The other one of the plurality of data structures is employed to locate and retrieve the plurality of related data objects from the data store. Additionally, in response to a request to delete at least one data object, each reference to each deleted data object in the plurality of data structures is automatically deleted.

In accordance with other aspects of the invention, a parent object is associated with each data object. Each parent object identifies each reference for the associated data object in the plurality of data structures. When a data object is deleted, the parent object associated with the deleted data object is employed to identify each reference for the deleted data object in the plurality of data structures and each reference is deleted.

In accordance with still other aspects of the invention, the data object can be a collector object that is associated with a member object that identifies one or more other data objects that are related to the collector object. The member object is employed to reference and retrieve each data object related to the collector object when the collector object is retrieved.

In accordance with yet other aspects of the invention, the plurality of related data objects have at least one related characteristic, including port, IP address and type. Also, the plurality of data structures may include List, Hash and Trie.

In accordance with other aspects of the invention, one of the plurality of data structures can be a Trie data structure that identifies a key in the request for a data object. The Trie data structure divides the key into segments and each segment is

employed to search the Trie data structure and locate a reference to the requested data object. The key can represent an IP address and/or a port. Also, each segment can be represented by at least one bit.

In accordance with still other aspects of the invention, the data store is a database. The database can be a relational, object-oriented, or some combination of the two, database. Also, the data store can be a data warehouse.

The invention may be implemented as a computer process, a computing system or as an article of manufacture such as a computer program product or computer readable medium. The computer program product may be a computer storage medium readable by a computer system and encoding a computer program of instructions for executing a computer process. The computer program product may also be a propagated signal on a carrier readable by a computing system and encoding a computer program of instructions for executing a computer process.

These and other features as well as advantages, which characterize the invention, will be apparent from a reading of the following detailed description and a review of the associated drawings.

Brief Description of the Drawings

FIGURE 1 illustrates a block diagram of a client-server operating environment used in an embodiment of the invention;

FIGURE 1B illustrates a block diagram of a client-server operating environment used with a server array controller in another embodiment of the invention;

FIGURE 2 shows a block diagram of a computing device used in an embodiment of the invention;

FIGURE 3 illustrates a block diagram of an overview of the complex data structure that includes a List data structure and a Trie data structure for one embodiment of the invention;

FIGURE 4A shows a block diagram of a data object;

FIGURE 4B illustrates a block diagram of a collector object;

FIGURE 5 shows a block diagram of nested collector and data objects;

FIGURE 6 illustrates a block diagram of an overview of the complex data structure with greater detail shown for the List data structure referencing data objects and collector objects;

FIGURE 7 shows a block diagram of an overview of the complex data structure with greater detail shown for the Trie data structure referencing data objects;

FIGURE 8 illustrates a block diagram of an overview of the complex data structure with greater detail shown for the Trie data structure referencing data objects and collector objects;

FIGURE 9A shows a block diagram of an IP address list;

FIGURE 9B illustrates a block diagram of a Port list;

FIGURE 10 shows a block diagram of an eight level Trie data structure shown in greater detail;

FIGURE 11 is a flow diagram illustrating the operational characteristics for determining one of a plurality of data structures best suited to reference and retrieve a requested data object; and

FIGURE 12 is a flow diagram illustrating the operational characteristics for deleting each reference to a deleted data object in each of the plurality of data structures.

Detailed Description of the Preferred Embodiment

An embodiment of the invention provides for employing a complex data structure to optimize the retrieval of data from a data store over a network. The complex data structure may include multiple data structures, e.g., List and Trie, which in parallel separately reference the same data objects in a data store. For a particular functional request to retrieve data, the complex data structure examines the request to determine whether it is associated the List or Trie data structures. In most cases, the Trie data structure is associated with a functional request for a single data object and the List data structure is associated with functional request to retrieve several related data objects.

Each data object is associated with a parent object that includes a list of every reference to the data object in both the Trie and List data structures. When a data object

is subsequently deleted, the complex data structure employs the parent object list to automatically delete every reference to the deleted data object in both the Trie and List data structures. Additionally, the complex data structure provides for a particular type of data object, i.e., a collector object, that is associated with a member object and which includes a list of other related data/collector objects that are referenced in a sub-tree below the node level of the collector object in the Trie data structure. Also, when the data associated with a collector object is requested, other data associated with the other data/collector objects on the member object list are automatically retrieved.

Another embodiment of the invention may employ a complex data structure that includes a List data structure and Hash data structure, which can be used with the List data structure to process functional requests for single objects. Generally, the Hash data structure requires less overhead, e.g., memory and processor cycles, than a Trie data structure to process a request for data. However, the Hash data structure would not support collector objects as discussed in greater detail below.

The logical operations of the various embodiments of the invention are implemented (1) as a sequence of computer implemented actions or program modules running on a computing device; and/or (2) as interconnected hardware or logic modules within the computing device. The implementation is a matter of choice dependent on the performance requirements of the computing system implementing the invention. Accordingly, the logical operations making up the embodiments of the invention described herein are referred to alternatively as operations, actions or modules. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Typically, the functionality of program modules may be combined or distributed in various embodiments.

FIGURE 1A illustrates one embodiment of an overview 10 of a suitable client-server operating environment in which the invention may be implemented. A client 12 is in communication with a server 16 over a network 14 such as the Internet. The server 16 is in communication with a data store 18 for retrieving data that may be requested by the client 12. In response to a request from the client 12 over the network

14, the server 16 can access the data store 18 and provide the requested data in an appropriate format to the client.

FIGURE 1B illustrates another embodiment of an overview 40 of an exemplary operating environment in which the invention may be implemented. A network device 42 is in communication with a server array controller 46 over a network 44 such as the Internet. The network device 42 may be a server, client, router, firewall, cache or another server array controller and the network 44 may also be a local area network or a wide area network for an enterprise. The server array controller 46 is in communication with a data store 48 that stores the address for at least one of a pool of virtual servers 52 managed by the server array controller 46. The virtual servers 52 are in communication over another network 50 with the server array controller 46, which manages the requests for data from the virtual servers. The server array controller 46 uses the information in the data store 48 to transform and direct traffic from one network device to another. The other network 50 may be a local area network, wide area network or the Internet.

FIGURE 2 illustrates one embodiment of a suitable operating environment for a computer 20 in which the invention may be implemented. The computer 20 is only one embodiment of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Other well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, client computers, laptop devices, multiprocessor devices, microprocessor based systems, programmable consumer electronics, network PCs, minicomputers, mainframe computers, client-server environments, distributed computing environments that include any of the above systems or devices, and the like.

FIGURE 2 shows functional components of the computer 20, including a processor system 28, a CPU 30, a memory system 22, a network interface 34, a computer readable media device 32 and an input/output device 36. The memory system 22 generally includes both volatile memory (e.g., RAM) and non-volatile memory (e.g., ROM, PC cards, etc.). An operating system 26 is resident in the memory system 22 and

executes on the CPU 30, such as Unix®, Linux®, or the Windows NT® operating system from the Microsoft ® Corporation.

One or more application programs 24 are loaded into the memory system 22 and run on the operating system 26. Examples of applications include email programs, scheduling programs, network management programs, PIM (personal information management) programs, word processing programs, spreadsheet programs, Internet browser programs, and so forth. The computer 20 has a power supply (not shown), which can be implemented as one or more batteries or might further include an external power source that overrides or recharges built-in batteries, such as an AC adapter.

The computer 20 may operate with at least some form of computer readable media. Computer readable media can be any available media that can be accessed by the computer readable media device 32. By way of example, and not limitation, computer readable media may comprise computer storage media and communications media. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules, or other data. Computer storage media includes, but is not limited to, random access memory (RAM), read only memory (ROM), electrically erasable programmable read only memory (EEPROM), flash memory or other memory technology, CD-ROM, digital versatile disk (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer 20. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, radio

frequency (RF), infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

FIGURE 3 shows an overview 100 of a complex data structure employed by the invention to provide optimal retrieval of requested data from a data store 108. A Functional Interface 102 processes each request for data. The functional interface is in communication with a List data structure 104 and a Trie data structure 106 and it determines which data structure matches the particular functional request for data. The Trie 104 and List 106 data structures provide parallel access to data stored in the data store 108. Although not shown, another embodiment of the invention may employ a Hash function data structure (instead of a Trie data structure) with the List data structure to optimize the retrieval of data in response to a functional request.

Objects in the data store 108 are retrieved/added/deleted through the Functional Interface 102. Retrieval functions are specific to either the Trie data structure 104 or the List data structure 106. For the Trie data structure 104, there is a Trie retrieval function that accepts a top level accessor object and a unique key and returns a container object, e.g., a data object or a collector object which are discussed in greater detail below. The List data structure 106 supports several retrieval functions, including mid level list accessor objects, changing the current state of a list, and retrieving at least one of the container objects that is a member of a list. Since the mid level list accessor objects reference associated top level Trie accessor object, they do not need to be specified by list functions.

A Trie retrieval function is generally employed to find a single container object when at least one of the Trie keys that give access to the object are known. For example, a Trie retrieval occurs when a data packet arrives at an appliance on an interface where a network address translation data store is not empty. In this example, a six byte source IP address and port can be used to attempt to retrieve information about a currently active network address translation connection. When the retrieval fails, the four byte source ip address can be used to attempt to find a network address translation initiator object which is stored as a collector object in the same complex data structure. If an initiator is found, a connection is added to the data store under the six byte source

IP address and port retrieval path. At the same time, a connection object will be automatically added to the initiator object's member list because the initiator object is a collector object with a retrieval path that is the prefix of the connection object's retrieval path.

The list retrieval functions are used whenever the appliance needs to sequentially access a group of container objects. For example, a connection in a network address translation data store can be deleted when it has not been used for a length of time. In this case, an appliance could periodically step through part of the list of all connection objects in the data store to determine whether some connections should be deleted.

In another example, both Trie and List functions may be used when a request is received for information about all of the network address translation connection objects that were created using an initiator object identified by a four byte ip address. In this case, the initiator object is retrieved using the Trie retrieval function and a four byte key. Then using the member list of the initiator object, the connection objects collected by the initiator are sequentially retrieved.

FIGURE 4A illustrates a data object 110 that includes a pointer 112 to the actual location of the requested data. A parent list (not shown) identifies a parent object 114 that is associated with the data object 110. The parent list identifies each parent object that references the data object in each data structure (List and Trie). When the data object 110 is subsequently deleted, the parent list is employed to automatically delete every parent object 114 in the List and Trie data structures that reference the deleted data object.

FIGURE 4B shows a collector object 116, which is similar in some ways to the data object 110 illustrated in FIGURE 4A, e.g., the collector object includes a data pointer 122 to the actual location of the requested data and a parent object 120 with a list identifying each reference to the collector object in both data structures. However, the collector object 116 further comprises a member object 118 with a list that identifies every other data object that is related to the collector object. When the data associated with the collector object 116 is retrieved, the invention employs the member object 118

to also automatically retrieve the data associated with the other related data objects on the member list.

Additionally, each collector object automatically adds all of the objects (data and/or collector) to its member object from the sub-tree (node levels) in the Trie data structure below the reference to the collector object. Thus, when a collector object is referenced by a Trie leaf node, the collector object will have an empty list in its member object.

In one embodiment of the invention, an add function automatically adds an object to the list for the member object of a collector object when the added object is added beneath the existing level of the collector object in a sub-tree of the Trie data structure. Additionally, when another add function is employed to add a new collector object to the Trie data structure at a level above the levels of previously existing objects in a sub-tree of the Trie data structure, these objects are automatically added to the list in the member object associated with the new collector object.

FIGURE 5 shows an overview 124 of one embodiment enabling a single collector object to reference other data objects and collector objects. A collector object 117 includes a member object (not shown) with a list that references all of the other data objects 110 and another collector object 119 disposed in a sub-tree below the reference to the collector object 117 in the Trie data structure. The other collector object 119 includes another member object (not shown) with another list that references the other data objects 110 disposed in another sub-tree below the reference to the collector object 119 in the Trie data structure. Thus, when the data associated with the collector object 117 is requested, other data associated with four other data objects 110 and another collector object 119 will be automatically retrieved.

FIGURE 6 illustrates an overview 126 that provides greater detail regarding the referencing of data/collector objects in a data store 109 by an exemplary List data structure 107. Although the Functional Interface 102 and the Trie data structure 104 are represented in FIGURE 6, their interaction with the data store 108 and the List data structure 107 is discussed in greater detail above and below. A Port list 130 includes references to three data objects 110 which are associated with a particular port ("Y"),

but have different IP addresses. FIGURE 9B shows an exemplary Port list 184 that includes references to three separate data objects with the same port ("Y") and different IP addresses ("F", "G", and "H").

Returning to FIGURE 6, an IP address list 128 references two data and one collector objects that are associated with different Ports, but have the same IP address ("X"). FIGURE 9A illustrates an exemplary IP address list 182 that includes references to three separate data objects with the same IP address ("X") and different Ports ("U", "V", and "W").

FIGURE 7 illustrates an overview 132 that provides greater detail regarding the referencing of data/collector objects in a data store 111 by an exemplary Trie data structure 103. Although the Functional Interface 102 and the List data structure 106 are represented in FIGURE 7, their interaction with the data store 111 and the Trie data structure 103 is discussed in greater detail above and below. Each transition between each node 136 in the twelve levels of the Trie data structure 103 is associated with a four bit key segment having the decimal range of 0 to 15. In combination, these key segments can serve as the reference to the data object 110 in the data store 111. In this example, each of the two exemplary paths through the twelve Trie nodes correspond to the same IP address and different ports (Port W and Port V) that are associated with the same data object 110 in the data store 111.

In greater detail, FIGURE 10 shows two paths through eight levels of Trie nodes 186 that correspond to two different IP addresses and which reference the same data object 188. The sequential combination of each transition value between each of the eight Trie nodes 186 corresponds to a particular IP address. At each Trie node, four bits are employed to indicate incoming and outgoing transition pointers associated with any of 16 possible values for each transition between each node. For this example, the sequential combination of the eight transition values represent two 32 IP bit addresses, as follows: (4,8,0,15,6,0,3,12,9) and (4,8,0,15,6,8,15,0,7).

FIGURE 8 shows an overview 144 that provides greater detail regarding the referencing of exemplary data/collector objects in a data store 115 by an exemplary Trie data structure 117. Although the Functional Interface 102 and the List data structure

104 are represented in FIGURE 8, they are discussed in greater detail above and below. Each transition between each node 136 in the path through the twelve levels in the Trie data structure 117 is associated with a four bit key segment. In this case, the three complete paths through all twelve levels of the Trie nodes reference separate data objects associated with different ports and/or IP addresses, i.e., data object 138 is associated with a different port and IP address and data objects 139 and 140 are associated with the same IP address and different ports.

Additionally, a collector object 142 is referenced by the transition values associated with a path through the first four levels of the Trie data structure and which represent the first four values in a 32 bit IP address. In this case, the collector object 142 has automatically included a member object (not shown) that lists a reference to each data object (139 and 140) that is disposed in the sub-tree (node levels) below the reference to the collector object 142. Since the first four key segments of the collector object 142 and the data objects 139 and 140 are the same, these data objects were automatically included in the member object for the collector object 142.

FIGURE 11 illustrates a flow chart 146 that shows the actions for one embodiment of the invention for retrieving data in response to a request. Moving from a start block, the operational flow advances to a block 148, where the Functional interfaces receives a function request for data at an IP address and port. The operational flow pushes to a decision block 150, where a determination is made whether the function request is a List function. If so, the operational flow advances to a block 162 where the List data structure is employed to locate a data object at the provided port and IP address and retrieve the requested data. The operational flow moves to a block 164, where when the data object is determined to be a "collector" object, a member object associated with the collector object is employed to retrieve other related data identified by other data objects that are referenced by the member object. Next, the operational flow moves to the End block and resumes calling program modules.

Alternatively, when the determination at the decision block 150 is false, the flow moves to a block 152 where key segments are created for a Trie data structure based on the port and IP address associated with the requested data. In one embodiment of the

invention, each key segment is four bits. In this example, four key segments are created for a 16 bit port and eight key segments are generated for a 32 bit IP address.

The operational flow moves to a block 154 where each key segment is employed to sequentially transition along a path through all of the node levels in the Trie data structure. Advancing to a decision block 156, a determination is made whether a null pointer value was found at any level in the path through the nodes of the Trie data structure. If true, the operational flow moves to an End block and resumes calling program modules.

Alternatively, when the determination at the decision block 156 is false, the operational flow advances to a block 158 where the complete path through the Trie data structure is employed to reference a data object and retrieve the requested data. The operational flow moves to a block 160 where when the referenced data object is determined to also be a collector object, a member list included with the collector object is employed to retrieve related data associated with other data objects on the member list. The operational flow transitions to the end block and resumes calling program modules.

FIGURE 12 illustrates a flow chart 166 that shows the actions for one embodiment of the invention for deleting data in response to a request. Moving from a start block, the flow advances to a block 168, where a function request is received by the Functional Interface to delete data associated with a particular port and IP address. The operational flow transitions to a decision block 170, where the Functional Interface determines whether the data to be deleted is a collector object. If so, the operational flow advances to a block 182 where the list of member objects is removed from the collector object. The operational flow moves to a block 172 where a parent object associated with the data object to be deleted is identified. A list may be associated with the object to be deleted that identifies multiple parent objects. Also, since a parent object is automatically generated for each data object, it is automatically updated until the associated data object is deleted.

Alternatively, when the object to be deleted is determined to not be a collector object at the decision block 170, the operational flow moves directly to the block 172 where substantially the same actions discussed above are performed.

Next, the operation flow advances to a decision block 174 where a determination is made whether the parent object is a trie node. If so, the operational flow moves to a block 184 where every trie node that references this parent object is deleted. The operational flow transitions to a decision block 178 where a determination is made whether more parent objects are included on the list associated with the data object to be deleted. If affirmative, the operational flow loops back to the block 172 and performs substantially the same actions as discussed above. However, when the determination at the decision block 178 is negative, the operational flow moves to a block 180 where the data object to be deleted is deallocated. The operational flow transitions to an end block and resumes calling other program modules.

Alternatively, if the determination at the decision block 174 is negative, i.e., the parent object is not a trie node, then the operational flow advances to a block 176 where the reference in the List data structure to the parent object is deleted. The operational flow moves to the decision block 178 where substantially the same actions discussed above are repeated.

One embodiment of the invention enables each data object in the data store to be accessed by one or more keys in constant time when the Trie data structure is employed to resolve a request for a data object. Each data object in the data store may be accessed by zero or more sequential lists. For one embodiment of the invention, an iterator object is employed to support multiple concurrent list states in the List data structure.

It is envisioned that the data store can be a relational, object oriented or a combination of the two, database. Also, the data space could be a data warehouse.

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.